SMART ARM-based Microcontrollers

AT13155: SAM L22 Segment Liquid Crystal Display(SLCD) Controller

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's SLCD functionality. The following driver API modes are covered by this manual:

- Polled APIs
- Callback APIs

The following peripheral is used by this module:

- Segment Liquid Crystal Display(SLCD)

The following devices can use this module:

- Atmel | SMART SAM L22

The outline of this documentation is as follows:

Table of Contents

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

3.1. Display Overview

A LCD display is made of several segments (pixels or complete symbols) which can be visible or invisible. A segment has two electrodes with liquid crystal between them. These electrodes are the common terminal (COM pin) and the segment terminal (SEG pin). When a voltage above a threshold voltage is applied across the liquid crystal, the segment becomes visible. The voltage must alternate, to avoid an electrophoresis effect in the liquid crystal, which degrades the display.

The LCD controller is intended for monochrome passive liquid crystal display (LCD) with up to 8 common terminals and up to 44 segment terminals. A charge pump provides LCD display supply which can be higher than supply voltage of the device. Each LCD pin, segment or common terminals, can be configured as general purpose I/O pins if not driven by LCD controller.

4. Special Considerations

4.1. I/O Lines

The SLCD pins (SEG and COM) are multiplexed with other peripherals. The user application must first configure the I/O controller, to give control of the requisite pins to the SLCD.

4.2. Power Management

The SLCD will continue to operate in any sleep mode where the selected source clock is running. The SLCD interrupts can be used to wake up the device from sleep modes. Events connected to the event system can trigger other operations in the system without exiting sleep modes.

The power consumption of SLCD itself can be minimized by:

- Using the lowest acceptable frame rate (refer to the LCD glass technical characteristics)
- Using the low-power waveform (default mode)
- Using automated modes of operation
- Configuring the lowest possible contrast value

5. Extra Information

For extra information, see Extra Information for SLCD. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for SLCD.

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type slcd_callback_t

```
typedef void(* slcd_callback_t )(enum slcd_callback_type type)
```

SLCD interrupt callback function type.

7.2. Structure Definitions

7.2.1. Struct slcd_automated_char_config

SLCD automated char configuration.

Table 7-1. Members

Type	Name	Description
uint8_t	com_line_num	Define the number of COM line per row, it equal to number of COM line - 1
uint32_t	data_mask	Segments data mask
uint8_t	digit_num	Define the number of digit, it must be greater than 1
enum slcd_frame_counter	fc	Frame counter selection for automated character mapping
enum slcd_automated_char_mode	mode	Display mode
enum slcd_automated_char_order	order	Mapping order in automated char mode
uint8_t	row_digit_num	Define the number of digit per row
uint8_t	scrolling_step	Define the number of steps in scrolling mode. scrolling_step = character string length - digit_num + 1
uint8_t	seg_line_num	Define the number of SEG line per digit, it equal to number of SEG line - 1
uint8_t	start_seg_line	Define the index of the first segment terminal of the digit to display

7.2.2. Struct slcd_blink_config

SLCD blink configuration.

Table 7-2. Members

Type	Name	Description
bool	blink_all_seg	All segments are allowed to blink if true, else only Selected segments are allowed to blink
enum slcd_frame_counter	fc	Frame counter selection for blinking

7.2.3. Struct slcd_circular_shift_config

SLCD circular shift configuration.

Table 7-3. Members

Type	Name	Description
uint16_t	data	Circular shift register value
enum slcd_circular_shift_dir	dir	Shift direction
enum slcd_frame_counter	fc	Frame counter selection for circular shift
uint8_t	size	Size of the circular shift register, MAX. size is 16

7.2.4. Struct slcd_config

Basic configuration for SLCDC.

Table 7-4. Members

Type	Name	Description
uint8_t	bias_buffer_duration	Bias buffer duration
bool	enable_bias_buffer	Enable bias buffer if true
bool	enable_ext_bias	Enable external bias capacitor if true
bool	enable_low_resistance	Enable Low resistance if true
uint8_t	low_resistance_duration	Low resistance network duration
bool	run_in_standby	Keep SLCD enabled in standby sleep mode if true
enum slcd_waveform_mode	waveform_mode	waveform mode selection

7.2.5. Struct slcd_events

Event flags for the SLCD module. This is used to enable and disable events via slcd_enable_events() and slcd_disable_events().

Table 7-5. Members

Type	Name	Description
bool	generate_event_on_fc0_overflow	Enable event generation on frame counter 0 overflow
bool	generate_event_on_fc1_overflow	Enable event generation on frame counter 1 overflow
bool	generate_event_on_fc2_overflow	Enable event generation on frame counter 2 overflow

7.3. Macro Definitions

7.3.1. Macro SLCD_CALLBACK_TYPE_NUM

```
#define SLCD_CALLBACK_TYPE_NUM
```

7.4. Function Definitions

7.4.1. SLCD Basic Operation Functions

7.4.1.1. Function slcd_get_config_defaults()

Initializes SLCD configurations struct to defaults.

```
void slcd_get_config_defaults(
        struct slcd_config * config)
```

Initailizes SLCD configuration struct to predefined safe default settings.

Table 7-6. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an instance of struct slcd_config

7.4.1.2. Function slcd_init()

Initialize SLCD module.

```
enum status_code slcd_init(
        struct slcd_config *const config)
```

Table 7-7. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an SLCD configuration structure

Note: SLCD cannot be Initialized while it is enabled.

Returns
Status of the configuration procedure.

Table 7-8. Return Values

Return value	Description
STATUS_OK	SLCD configuration went successful
STATUS_ERR_INVALID_ARG	If an invalid configuration was supplied

7.4.1.3. Function slcd_enable()

Enables the SLCD module.

```
void slcd_enable( void )
```

Enables the SLCD module once it has been configured, ready for use. Most module configuration parameters cannot be altered while the module is enabled.

7.4.1.4. Function slcd_disable()

Disables the SLCD module.

```
void slcd_disable( void )
```

Disables the SLCD module.

7.4.1.5. Function slcd_is_enabled()

Check if SLCD module is enabled or not.

```
bool slcd_is_enabled( void )
```

Check if SLCD module is enabled or not.

Returns
Enable status.

Table 7-9. Return Values

Return value	Description
true	SLCD module is enabled
false	SLCD module is disabled

7.4.1.6. Function slcd_reset()

Reset the SLCD module.

```
void slcd_reset( void )
```

Reset the SLCD module.

7.4.1.7. Function slcd_set_contrast()

Set the SLCD fine contrast.

```
enum status_code slcd_set_contrast(
        uint8_t contrast)
```

The LCD contrast is defined by the value of VLCD voltage. The higher is the VLCD voltage, the higher is the contrast. The software contrast adjustment is only possible in internal supply mode. In internal supply mode, VLCD is in the range 2.5V to 3.5V. VLCD can be adjusted with 16 steps, each step is 60 mV.The contrast value can be written at any time.

Table 7-10. Parameters

Data direction	Parameter name	Description
[in]	contrast	Contrast value

Returns
Status of set contrast.

Table 7-11. Return Values

Return value	Description
STATUS_OK	SLCD contrast set successful
STATUS_ERR_INVALID_ARG	SLCD is not working in internal supply mode

7.4.1.8. Function slcd_is_syncing()

Determines if SLCD module is currently synchronizing to the bus.

```
bool slcd_is_syncing( void )
```

Checks to see if the underlying hardware peripheral module(s) are currently synchronizing across multiple clock domains to the hardware bus, This function can be used to delay further operations on a module until such time that it is ready, to prevent blocking delays for synchronization in the user application.

Returns
Synchronization status of the underlying hardware module.

Table 7-12. Return Values

Return value	Description
true	If the module synchronization is ongoing
false	If the module has completed synchronization

7.4.1.9. Function slcd_lock_shadow_memory()

Lock shadow memory.

```
void slcd_lock_shadow_memory( void )
```

It allows update of shadow display memory. If the display memory is modified, the display remains unchanged when locked.

7.4.1.10. Function slcd_unlock_shadow_memory()

Unlock shadow memory.

```
void slcd_unlock_shadow_memory( void )
```

Unlock the shadow display memory.

7.4.1.11. Function slcd_clear_display_memory()

Clear display memory.

```
void slcd_clear_display_memory( void )
```

Clears immediately the display memory.

7.4.1.12. Function slcd_enable_display()

Display enable.

```
void slcd_enable_display( void )
```

Enable COM/SEG signal output.

7.4.1.13. Function slcd_disable_display()

Display disable.

```
void slcd_disable_display( void )
```

Disable COM/SEG signal output.

7.4.1.14. Function slcd_dma_display_memory_update_fc_sel()

DMA display memory update frame counter selection.

```
void slcd_dma_display_memory_update_fc_sel(
        enum slcd_frame_counter fc)
```

It's used to select the frame counter for DMA to update the display memory.

Note: It can be called only before the module is enabled.

Table 7-13. Parameters

Data direction	Parameter name	Description
[in]	fc	Frame coungter index

7.4.2. SLCD Blink Functions

7.4.2.1. Function slcd_enable_blink()

Blink mode enable.

```
void slcd_enable_blink( void )
```

Enable blink mode.

7.4.2.2. Function slcd_disable_blink()

Blink mode disable.

```
void slcd_disable_blink( void )
```

Disable blink mode.

7.4.2.3. Function slcd_blink_get_config_defaults()

Initializes SLCD blink configurations struct to defaults.

```
void slcd_blink_get_config_defaults(
        struct slcd_blink_config * blink_config)
```

Initailizes SLCD blink configuration struct to predefined safe default settings.

Table 7-14. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an instance of struct slcd_blink_config

7.4.2.4. Function slcd_blink_set_config()

Set SLCD blink mode.

```
enum status_code slcd_blink_set_config(
        struct slcd_blink_config *const blink_config)
```

Set SLCD blink mode.

Note: SLCD blink cannot be set while module or blink is enabled.

Table 7-15. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an SLCD blink configuration structure

Returns
Status of the configuration procedure.

Table 7-16. Return Values

Return value	Description
STATUS_OK	SLCD blink configuration went successful
STATUS_ERR_INVALID_ARG	If blink configuration failed

7.4.2.5. Function slcd_clear_blink_all_pixel()

Stop all SLCD pixels/segments from blinking.

```
void slcd_clear_blink_all_pixel( void )
```

7.4.2.6. Function slcd_clear_blink_pixel()

Stop a specified SLCD pixel/segment from blinking.

```
void slcd_clear_blink_pixel(
        uint8_t pix_com,
        uint8_t pix_seg)
```

Table 7-17. Parameters

Data direction	Parameter name	Description
[in]	pix_com	Pixel/segment COM coordinate
[in]	pix_seg	Pixel/segment SEG coordinate (range 0 to 1 inclusive)

7.4.2.7. Function slcd_set_blink_pixel()

Start an SLCD pixel/segment blinking.

```
void slcd_set_blink_pixel(
        uint8_t pix_com,
        uint8_t pix_seg)
```

Table 7-18. Parameters

Data direction	Parameter name	Description
[in]	pix_com	Pixel/segment COM coordinate
[in]	pix_seg	Pixel/segment SEG coordinate (range 0 to 1 inclusive)

7.4.3. SLCD Blank Functions

7.4.3.1. Function slcd_enable_blank()

Blank mode enable.

```
void slcd_enable_blank( void )
```

Enable blank mode.

7.4.3.2. Function slcd_disable_blank()

Blank mode disable.

```
void slcd_disable_blank( void )
```

Disable blank mode.

7.4.4. SLCD Event Functions

7.4.4.1. Function slcd_enable_events()

Enables a SLCD event output.

```
void slcd_enable_events(
        struct slcd_events *const events)
```

Enables one or more output events.

Note: Events cannot be altered while the module is enabled.

Table 7-19. Parameters

Data direction	Parameter name	Description
[in]	events	Struct containing flags of events to enable

7.4.4.2. Function slcd_disable_events()

Disables a SLCD event output.

```
void slcd_disable_events(
        struct slcd_events *const events)
```

Disables one or more SLCD events output.

Table 7-20. Parameters

Data direction	Parameter name	Description
[in]	events	Struct containing flags of events to disable

7.4.5. SLCD Frame Counter Functions

7.4.5.1. Function slcd_set_frame_counter()

Frame counter configuration.

```
void slcd_set_frame_counter(
        enum slcd_frame_counter fc,
        bool presc_bypass_enable,
        uint16_t overflow_value)
```

Config frame counter.

Note: Frame counter cannot be set while it is enabled.

Table 7-21. Parameters

Data direction	Parameter name	Description
[in]	fc	Frame counter index
[in]	presc_bypass_enable	Bypass of the frame counter prescaler
[in]	overflow_value	Frame counter overflow value. The number of frame before overflow is ((overflow_value+1)*8) when presc_bypass_enable=0 else (overflow_value+1). The MAX. overflow value is 0x1FFFF.

7.4.5.2. Function slcd_enable_frame_counter()

Enables a frame counter.

```
void slcd_enable_frame_counter(
        enum slcd_frame_counter fc)
```

Enables one frame counter.

Table 7-22. Parameters

Data direction	Parameter name	Description
[in]	fc	Frame counter index

7.4.5.3. Function slcd_disable_frame_counter()

Disable a frame counter.

```
void slcd_disable_frame_counter(
        enum slcd_frame_counter fc)
```

Disable one frame counter.

Table 7-23. Parameters

Data direction	Parameter name	Description
[in]	fc	Frame counter index

7.4.6. Display Memory Functions

CPU can access display memory in direct access or in indirect access.

7.4.6.1. Function slcd_set_display_memory()

Set all bits in the SLCD display memory high.

```
void slcd_set_display_memory( void )
```

7.4.6.2. Function slcd_set_pixel()

Enable the specified pixel/segment in the SLCD display memory.

```
void slcd_set_pixel(
        uint8_t pix_com,
        uint8_t pix_seg)
```

Table 7-24. Parameters

Data direction	Parameter name	Description
[in]	pix_com	Pixel/segment COM coordinate,within [0-7]
[in]	pix_seg	Pixel/segment SEG coordinate within [0-43]

7.4.6.3. Function slcd_clear_pixel()

Disable the specified pixel/segment in the SLCD display memory.

```
void slcd_clear_pixel(
        uint8_t pix_com,
        uint8_t pix_seg)
```

Table 7-25. Parameters

Data direction	Parameter name	Description
[in]	pix_com	Pixel/segment COM coordinate
[in]	pix_seg	Pixel/segment SEG coordinate

7.4.6.4. Function slcd_set_seg_data()

Set the specified segment in the SLCD display memory.

```
void slcd_set_seg_data(
        uint8_t seg_data,
        uint8_t byte_offset,
        uint8_t seg_mask)
```

Table 7-26. Parameters

Data direction	Parameter name	Description
[in]	pix_seg	Pixel/segment SEG coordinate
[in]	byte_offset	Byte offset in display memory
[in]	seg_mask	Byte offset in display memory

7.4.7. Character Mapping Functions

7.4.7.1. Function slcd_character_map_set()

Set SLCD character mapping.

```
void slcd_character_map_set(
        enum slcd_automated_char_order order,
        uint8_t seg_line_num)
```

Set Character mode amd SEG line per digit.

Table 7-27. Parameters

Data direction	Parameter name	Description
[in]	order	Mapping order in char mode
[in]	seg_line_num	Define the number of SEG line per digit, it equal to number of SEG line - 1

7.4.7.2. Function slcd_character_write_data()

Write segments data to display memory in character mode.

```
void slcd_character_write_data(
        uint8_t com_line_index,
        uint8_t seg_line_index,
        uint32_t seg_data,
        uint32_t data_mask)
```

Table 7-28. Parameters

Data direction	Parameter name	Description
[in]	seg_data	Pixel/segment data
[in]	data_mask	Segments data mask

Data direction	Parameter name	Description
[in]	com_line_index	COM line index
[in]	seg_line_index	Segments line index

7.4.7.3. Function slcd_enable_automated_character()

Enables automated character display.

```
void slcd_enable_automated_character( void )
```

Enables automated character display.

7.4.7.4. Function slcd_disable_automated_character()

Disables automated character display.

```
void slcd_disable_automated_character( void )
```

Disables automated character display.

7.4.7.5. Function slcd_automated_char_get_config_default()

Initializes SLCD Automated Character configurations struct to defaults.

```
void slcd_automated_char_get_config_default(
        struct slcd_automated_char_config * config)
```

Initailizes SLCD Automated Character configuration struct to predefined safe default settings.

Table 7-29. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an instance of struct slcd_automated_char_config

7.4.7.6. Function slcd_automated_char_set_config()

Set SLCD automated character.

```
enum status_code slcd_automated_char_set_config(
        struct slcd_automated_char_config *const config)
```

Set automated character mode.

Note: SLCD automated character mode cannot be set while module or automated character is enabled.

Table 7-30. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an SLCD automated character configuration structure

Returns
Status of the configuration procedure.

Table 7-31. Return Values

Return value	Description
STATUS_OK	SLCD automated character configuration went successful
STATUS_ERR_INVALID_ARG	If automated character configuration failed

7.4.8. Automated Bit Mapping Functions

7.4.8.1. Function slcd_enable_automated_bit()

Enables automated bit display.

```
void slcd_enable_automated_bit( void )
```

Enables automated bit display.

7.4.8.2. Function slcd_disable_automated_bit()

Disables automated bit display.

```
void slcd_disable_automated_bit( void )
```

Disables automated bit display.

7.4.8.3. Function slcd_set_automated_bit()

Sets automated bit display.

```
void slcd_set_automated_bit(
        uint8_t size,
        enum slcd_frame_counter fc)
```

Sets automated bit display.

Note: Automated bit cannot be set while it is enabled or busy.

7.4.9. Autonomous Segment Animation

7.4.9.1. Function slcd_enable_circular_shift()

Enable SLCD circular shift mode.

```
void slcd_enable_circular_shift( void )
```

7.4.9.2. Function slcd_disable_circular_shift()

Disable SLCD circular shift mode.

```
void slcd_disable_circular_shift( void )
```

7.4.9.3. Function slcd_circular_shift_get_config_defaults()

Initializes circular shift configurations struct to defaults.

```
void slcd_circular_shift_get_config_defaults(
        struct slcd_circular_shift_config *const config)
```

Initailizes circular shift configuration struct to predefined safe default settings.

Table 7-32. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an instance of struct slcd_circular_shift_config

7.4.9.4. Function slcd_circular_shift_set_config()

Set SLCD circular shift.

```
enum status_code slcd_circular_shift_set_config(
        struct slcd_circular_shift_config *const config)
```

Set circular shift mode.

Note: SLCD circular shift mode cannot be set while module or circular shift is enabled.

Table 7-33. Parameters

Data direction	Parameter name	Description
[in]	config	Pointer to an SLCD circular shift configuration structure

Returns
Status of the configuration procedure.

Table 7-34. Return Values

Return value	Description
STATUS_OK	SLCD circular shift configuration went successful
STATUS_ERR_INVALID_ARG	If circular shift configuration failed

7.4.10. SLCD Status

7.4.10.1. Function slcd_get_auto_bit_status()

Checks if auto bit mapping state machine is busy.

```
bool slcd_get_auto_bit_status( void )
```

Checks if auto bit mapping state machine is busy or not.

Table 7-35. Return Values

Return value	Description
true	Auto bit mapping state machine is busy
false	Auto bit mapping state machine is idle

7.4.10.2. Function slcd_get_auto_char_status()

Checks if auto character mapping state machine is busy.

```
bool slcd_get_auto_char_status( void )
```

Checks if auto character state machine is busy or not.

Table 7-36. Return Values

Return value	Description
true	Auto character mapping state machine is busy
false	Auto character mapping state machine is idle

7.4.10.3. Function slcd_get_char_writing_status()

Checks if character writing function is busy.

```
bool slcd_get_char_writing_status( void )
```

Checksif character writing function is busy or not.

Table 7-37. Return Values

Return value	Description
true	Character writting function is busy
false	Character writting function is ready for use

7.4.10.4. Function slcd_get_vlcd_vdd33_status()

Checks VLCD and VDD33 status.

```
bool slcd_get_vlcd_vdd33_status( void )
```

Checks VLCD and VDD33 status.

Table 7-38. Return Values

Return value	Description
true	VDD33 is greater than target VLCD
false	Target VLCD is greater than vdd33

7.4.10.5. Function slcd_get_charge_pump_status()

Checks LCD charge pump status.

```
bool slcd_get_charge_pump_status( void )
```

Checks LCD Charge Pump Status.

Table 7-39. Return Values

Return value	Description
true	LCD power charge pump is running
false	LCD power charge pump is stopped

7.4.10.6. Function slcd_get_vlcd_ready_status()

Checks if VLCD is ready.

```
bool slcd_get_vlcd_ready_status( void )
```

Checks if VLCD is well regulated to the target value.

Table 7-40. Return Values

Return value	Description
true	VLCD is well regulated to the target value
false	VLCD is not well regulated to the target value

7.4.11. Callback Function

7.4.11.1. Function slcd_register_callback()

Registers a callback.

```
enum status_code slcd_register_callback(
        const slcd_callback_t callback,
        const enum slcd_callback_type type)
```

Registers a callback function which is implemented by the user.

Note: The callback must be enabled by slcd_enable_callback, in order for the interrupt handler to call it when the conditions for the callback type is met.

Table 7-41. Parameters

Data direction	Parameter name	Description
[in]	callback_func	Pointer to callback function
[in]	callback_type	Callback type given by an enum

Table 7-42. Return Values

Return value	Description
STATUS_OK	The function exited successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied

7.4.11.2. Function slcd_unregister_callback()

Unregisters a callback.

```
enum status_code slcd_unregister_callback(
        const slcd_callback_t callback,
        const enum slcd_callback_type type)
```

Unregisters a callback function implemented by the user.

Table 7-43. Parameters

Data direction	Parameter name	Description
[in]	callback_type	Callback type given by an enum

Table 7-44. Return Values

Return value	Description
STATUS_OK	The function exited successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied

7.4.11.3. Function slcd_enable_callback()

Enable an SLCD callback.

```
void slcd_enable_callback(
        const enum slcd_callback_type type)
```

Table 7-45. Parameters

Data direction	Parameter name	Description
[in]	type	Callback source type

Table 7-46. Return Values

Return value	Description
STATUS_OK	The function exited successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied

7.4.11.4. Function slcd_disable_callback()

Disable an SLCD callback.

```
void slcd_disable_callback(
        const enum slcd_callback_type type)
```

Table 7-47. Parameters

Data direction	Parameter name	Description
[in]	type	Callback source type

Table 7-48. Return Values

Return value	Description
STATUS_OK	The function exited successfully
STATUS_ERR_INVALID_ARG	If an invalid callback type was supplied

7.5. Enumeration Definitions

7.5.1. Enum slcd_automated_char_mode

Enum automated char display mode.

Table 7-49. Members

Enum value	Description
SLCD_AUTOMATED_CHAR_SEQ	Sequential Display Mode
SLCD_AUTOMATED_CHAR_SCROLL	Scrolling Display Mode

7.5.2. Enum slcd_automated_char_order

Enum automated char order.

Table 7-50. Members

Enum value	Description
SLCD_AUTOMATED_CHAR_START_FROM_BOTTOM_RIGHT	Segment is starting from bottom right
SLCD_AUTOMATED_CHAR_START_FROM_BOTTOM_LEFT	Segment is starting from bottom left

7.5.3. Enum slcd_callback_type

Enum SLCD callback type.

Table 7-51. Members

Enum value	Description
SLCD_CALLBACK_FC0_OVERFLOW	Frame Counter 0 Overflow callback
SLCD_CALLBACK_FC1_OVERFLOW	Frame Counter 1 Overflow callback
SLCD_CALLBACK_FC2_OVERFLOW	Frame Counter 2 Overflow callback
SLCD_CALLBACK_VLCD_READY	VLCD Ready Toggle callback
SLCD_CALLBACK_VLCD_TOGGLE	VLCD Status Toggle callback
SLCD_CALLBACK_PUMP_TOGGLE	Pump Run Status Toggle callback

7.5.4. Enum slcd_circular_shift_dir

Enum SLCD circular shift direction.

Table 7-52. Members

Enum value	Description
SLCD_CIRCULAR_SHIFT_LEFT	Circular shift direction is left
SLCD_CIRCULAR_SHIFT_RIGHT	Circular shift direction is right

7.5.5. Enum slcd_frame_counter

Enum SLCD frame counter definition.

Table 7-53. Members

Enum value	Description
SLCD_FRAME_COUNTER_0	SLCD frame counter 0
SLCD_FRAME_COUNTER_1	SLCD frame counter 1
SLCD_FRAME_COUNTER_2	SLCD frame counter 2

7.5.6. Enum slcd_waveform_mode

Enum waveform mode.

Table 7-54. Members

Enum value	Description
SLCD_LOW_POWER_WAVEFORM_MODE	Low power waveform mode
SLCD_STANDARD_WAVEFORM_MODE	Standard waveform mode

8. Extra Information for SLCD

8.1. Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
SLCD	Segment Liquid Crystal Display
COM	Common, denotes how many segments are connected to a segment terminal
SEG	Segment, the least viewing element (pixel) which can be on or off
Duty	1/(Number of common terminals on an actual LCD display)
Bias	1/(Number of voltage levels used driving a LCD display -1)
Frame Rate	Number of times the LCD segments are energized per second

8.2. Dependencies

This driver has the following dependencies:

- None

8.3. Errata

There are no errata related to this driver.

8.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial release

9. Examples for SLCD

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM Segment Liquid Crystal Display(SLCD) Controller. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for SLCD

9.1. Quick Start Guide for SLCD

The supported board list:

- SAM L22 Xplained Pro

The SEGMENT LCD1 Xplained Pro extension board must be connected to extension header 5 on the SAM L22 Xplained Pro.

This example demonstrates how to use the SLCD driver, it covers the following cases:

- Display Memory Mapping(Direct Access and Indirect Access)
- Character Mapping
- Blinking
- Automated Character Mapping
- Automated Bit Mapping

Upon startup, the program uses the USART driver to display application output message.

9.1.1. Quick Start

9.1.1.1. Prerequisites

There are no prerequisites for this use case.

9.1.1.2. Code

Add to the main application source file, outside of any functions:

```
/*Character map 0-9,A-Z*/
const uint32_t character_map[] = {
    0x2e74,0x440,0x23c4,0x25c4,0x5e0,0x25a4,0x27a4,0x444,0x27e4,0x25e4, /
*0-9*/
    0x7e4,0xa545,0x2224,0xa445,0x23a4,0x3a4,0x2724, /*A-G*/
    0x7e0,0xa005,0x2640,0x12b0,0x2220,0x678,0x1668, /*H-N*/
    0x2664,0x3e4,0x3664,0x13e4,0x25a4,0x8005,/*O-T*/
    0x2660,0xa30,0x1e60,0x1818,0x8018,0x2814/*U-Z*/
};

/* HELLO ATMEL map */
uint32_t display_array[11]={0x7e0,0x23a4,0x2220,0x2220,0x2664,0,
                            0x7e4,0x8005,0x678,0x23a4,0x2220};
#define DISPLAY_ARRAY_SIZE (sizeof(display_array)/sizeof(uint32_t))

/*ICON COM/SEG map */
#define C42412A_ICON_USB            1, 1
#define C42412A_ICON_BAT            0, 0
#define C42412A_ICON_ATMEL          0, 1

/* Automated Bit Mapping
```

```
 Format: [00,offset,data mask,data]
 data:[7-0]
 data mask:[15-8]
 offset:[21-16]
*/
const uint32_t abm_matrix_string0[] = {
    0x00002f10,0x00012211,0x00022211, /* COM0  */
    0x00060f00,0x00070000,0x00080000, /* COM1  */
    0x000c0f80,0x000d0088,0x000e0088, /* COM2 */
    0x00124f10,0x00134411,0x00144411  /* COM3 */
 };

const uint32_t abm_matrix_string1[] = {
    0x00002f10,0x00012211,0x00022211, /* COM0  */
    0x00060f80,0x00070088,0x00080088, /* COM1  */
    0x000c0f90,0x000d0099,0x000e0099, /* COM2  */
    0x00124f10,0x00134411,0x00144411  /* COM3  */
 };

const uint32_t abm_matrix_string2[] = {
    0x00002f90,0x00012299,0x00022299, /* COM0  */
    0x00060f10,0x00070011,0x00080011, /* COM1  */
    0x000c0f80,0x000d0088,0x000e0088, /* COM2  */
    0x00124f10,0x00134411,0x00144411  /* COM3  */
 };

const uint32_t abm_matrix_string3[] = {
    0x00002f10,0x00012211,0x00022211, /* COM0  */
    0x00060f00,0x00070000,0x00080000, /* COM1  */
    0x000c0f00,0x000d0000,0x000e0000, /* COM2  */
    0x00124f80,0x00134488,0x00144488  /* COM3  */
 };

#define ABM_MATRIX_SIZE (sizeof(abm_matrix_string0)/sizeof(uint32_t))
```

Add to the main application source file, outside of any functions:

```
struct dma_resource example_resource_abm;
struct dma_resource example_resource_acm;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_acm ;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm1;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm2;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm3;

struct usart_module usart_instance;
```

Copy-paste the following setup code to your user application:

```
static void configure_console(void)
{
    struct usart_config config_usart;
    usart_get_config_defaults(&config_usart);
    config_usart.baudrate    = 38400;
    config_usart.mux_setting = EDBG_CDC_SERCOM_MUX_SETTING;
    config_usart.pinmux_pad0 = EDBG_CDC_SERCOM_PINMUX_PAD0;
    config_usart.pinmux_pad1 = EDBG_CDC_SERCOM_PINMUX_PAD1;
    config_usart.pinmux_pad2 = EDBG_CDC_SERCOM_PINMUX_PAD2;
```

```
    config_usart.pinmux_pad3 = EDBG_CDC_SERCOM_PINMUX_PAD3;
    stdio_serial_init(&usart_instance, EDBG_CDC_MODULE, &config_usart);
    usart_enable(&usart_instance);
}
static void configure_dma_acm(void)
{
    struct dma_resource_config acm_config;

    dma_get_config_defaults(&acm_config);

    acm_config.peripheral_trigger = SLCD_DMAC_ID_ACMDRDY;
    acm_config.trigger_action = DMA_TRIGGER_ACTON_BEAT;

    dma_allocate(&example_resource_acm, &acm_config);

    struct dma_descriptor_config acm_descriptor_config;

    dma_descriptor_get_config_defaults(&acm_descriptor_config);

    acm_descriptor_config.beat_size = DMA_BEAT_SIZE_WORD;
    acm_descriptor_config.src_increment_enable = true;
    acm_descriptor_config.block_transfer_count = DISPLAY_ARRAY_SIZE;
    acm_descriptor_config.step_selection = DMA_STEPSEL_SRC;
    acm_descriptor_config.dst_increment_enable = false;
    acm_descriptor_config.source_address = (uint32_t)display_array
                                           + sizeof(display_array);
    acm_descriptor_config.destination_address = (uint32_t)&SLCD-
>CMDATA.reg;
    acm_descriptor_config.next_descriptor_address =
(uint32_t)&example_descriptor_acm;

    dma_descriptor_create(&example_descriptor_acm, &acm_descriptor_config);

    dma_add_descriptor(&example_resource_acm, &example_descriptor_acm);
    dma_start_transfer_job(&example_resource_acm);
}

static void configure_dma_abm(void)
{

    struct dma_resource_config abm_config;
    dma_get_config_defaults(&abm_config);
    abm_config.peripheral_trigger = SLCD_DMAC_ID_ABMDRDY;
    abm_config.trigger_action = DMA_TRIGGER_ACTON_BEAT;

    dma_allocate(&example_resource_abm, &abm_config);

    struct dma_descriptor_config abm_descriptor_config;
    dma_descriptor_get_config_defaults(&abm_descriptor_config);

    abm_descriptor_config.beat_size = DMA_BEAT_SIZE_WORD;
    abm_descriptor_config.src_increment_enable = true;
    abm_descriptor_config.step_selection = DMA_STEPSEL_SRC;
    abm_descriptor_config.dst_increment_enable = false;
    abm_descriptor_config.block_transfer_count = ABM_MATRIX_SIZE;
    abm_descriptor_config.source_address = (uint32_t)abm_matrix_string0
                                           + sizeof(abm_matrix_string0);
    abm_descriptor_config.destination_address = (uint32_t)&SLCD-
>ISDATA.reg;
    abm_descriptor_config.next_descriptor_address =
(uint32_t)&example_descriptor_abm1;
```

```
    dma_descriptor_create(&example_descriptor_abm, &abm_descriptor_config);

    abm_descriptor_config.source_address = (uint32_t)abm_matrix_string1
                                           + sizeof(abm_matrix_string1);
    abm_descriptor_config.next_descriptor_address =
(uint32_t)&example_descriptor_abm2;
    dma_descriptor_create(&example_descriptor_abm1,
&abm_descriptor_config);

    abm_descriptor_config.source_address = (uint32_t)abm_matrix_string2
                                           + sizeof(abm_matrix_string2);
    abm_descriptor_config.next_descriptor_address =
(uint32_t)&example_descriptor_abm3;
    dma_descriptor_create(&example_descriptor_abm2,
&abm_descriptor_config);

    abm_descriptor_config.source_address = (uint32_t)abm_matrix_string3
                                           + sizeof(abm_matrix_string3);
    abm_descriptor_config.next_descriptor_address =
(uint32_t)&example_descriptor_abm;
    dma_descriptor_create(&example_descriptor_abm3,
&abm_descriptor_config);

    dma_add_descriptor(&example_resource_abm, &example_descriptor_abm);
    dma_start_transfer_job(&example_resource_abm);
}
```

Add to user application initialization (typically the start of `main()`):

```
    struct slcd_config config;
    system_init();
    configure_console();
    delay_init();

    /* Turn on the backlight. */
    port_pin_set_output_level(SLCD_BACLKLIGHT,true);

    printf("SLCD example starts\r\n");
    slcd_get_config_defaults(&config);
    slcd_init(&config);
    slcd_set_contrast(0x8);
    configure_dma_acm();
    configure_dma_abm();

    slcd_enable();
```

9.1.1.3. Workflow

1. Define charactor map data.

```
/*Character map 0-9,A-Z*/
const uint32_t character_map[] = {

0x2e74,0x440,0x23c4,0x25c4,0x5e0,0x25a4,0x27a4,0x444,0x27e4,0x25e4, /
*0-9*/
    0x7e4,0xa545,0x2224,0xa445,0x23a4,0x3a4,0x2724, /*A-G*/
    0x7e0,0xa005,0x2640,0x12b0,0x2220,0x678,0x1668, /*H-N*/
    0x2664,0x3e4,0x3664,0x13e4,0x25a4,0x8005,/*O-T*/
    0x2660,0xa30,0x1e60,0x1818,0x8018,0x2814/*U-Z*/
};
```

```
/* HELLO ATMEL map */
uint32_t display_array[11]={0x7e0,0x23a4,0x2220,0x2220,0x2664,0,
                            0x7e4,0x8005,0x678,0x23a4,0x2220};
#define DISPLAY_ARRAY_SIZE (sizeof(display_array)/sizeof(uint32_t))

/*ICON COM/SEG map */
#define C42412A_ICON_USB            1, 1
#define C42412A_ICON_BAT            0, 0
#define C42412A_ICON_ATMEL          0, 1

/* Automated Bit Mapping
 Format: [00,offset,data mask,data]
 data:[7-0]
 data mask:[15-8]
 offset:[21-16]
*/
const uint32_t abm_matrix_string0[] = {
    0x00002f10,0x00012211,0x00022211, /* COM0  */
    0x00060f00,0x00070000,0x00080000, /* COM1  */
    0x000c0f80,0x000d0088,0x000e0088, /* COM2 */
    0x00124f10,0x00134411,0x00144411  /* COM3 */
 };

const uint32_t abm_matrix_string1[] = {
    0x00002f10,0x00012211,0x00022211, /* COM0  */
    0x00060f80,0x00070088,0x00080088, /* COM1  */
    0x000c0f90,0x000d0099,0x000e0099, /* COM2  */
    0x00124f10,0x00134411,0x00144411  /* COM3  */
 };

const uint32_t abm_matrix_string2[] = {
    0x00002f90,0x00012299,0x00022299, /* COM0  */
    0x00060f10,0x00070011,0x00080011, /* COM1  */
    0x000c0f80,0x000d0088,0x000e0088, /* COM2  */
    0x00124f10,0x00134411,0x00144411  /* COM3  */
 };

const uint32_t abm_matrix_string3[] = {
    0x00002f10,0x00012211,0x00022211, /* COM0  */
    0x00060f00,0x00070000,0x00080000, /* COM1  */
    0x000c0f00,0x000d0000,0x000e0000, /* COM2  */
    0x00124f80,0x00134488,0x00144488  /* COM3  */
 };

#define ABM_MATRIX_SIZE (sizeof(abm_matrix_string0)/sizeof(uint32_t))
```

2. Create related module variable and software instance structure.

```
struct dma_resource example_resource_abm;
struct dma_resource example_resource_acm;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_acm ;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm1;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm2;
COMPILER_ALIGNED(16)
DmacDescriptor example_descriptor_abm3;

struct usart_module usart_instance;
```

3. Configure, initialize, and enable slcd module.
    1. Configuration slcd struct, which can be filled out to adjust the configuration of a physical slcd peripheral.

```
slcd_get_config_defaults(&config);
slcd_init(&config);
slcd_set_contrast(0x8);
```

    2. Config DMA module for automated character mapping and automated bit mapping.

```
configure_dma_acm();
configure_dma_abm();
```

    3. Enable the slcd module.

```
slcd_enable();
```

9.1.2. Use Case

9.1.2.1. Code

Copy-paste the following code to your user application:

```
/* 1. Display all*/
slcd_set_display_memory();
delay_s(1);
slcd_clear_display_memory();

/* 2. Display icon*/
slcd_set_pixel(C42412A_ICON_USB);
slcd_set_pixel(C42412A_ICON_BAT);
slcd_set_pixel(C42412A_ICON_ATMEL);
delay_s(1);

/* 3. Character map*/
slcd_character_map_set(SLCD_AUTOMATED_CHAR_START_FROM_BOTTOM_RIGHT,3);
for(uint32_t i = 0 ; i < 5 ; i++) {
    slcd_character_write_data(0,4+i*4,character_map[10+i],0xFF4002);
}
delay_s(2);

/* 4. Blinking*/
slcd_disable();
struct slcd_blink_config blink_config;
slcd_blink_get_config_defaults(&blink_config);
blink_config.blink_all_seg = false;
slcd_blink_set_config(&blink_config);

for(uint32_t i=0; i<4; i++){
    slcd_set_blink_pixel(i,0);
    slcd_set_blink_pixel(i,1);
}

slcd_set_frame_counter(SLCD_FRAME_COUNTER_0,true,0x18);
slcd_enable_frame_counter(SLCD_FRAME_COUNTER_0);

slcd_enable_blink();
slcd_enable();
delay_s(2);

/* 5. Automated Character Mapping*/
slcd_disable();
struct slcd_automated_char_config acm_config;
```

```
slcd_automated_char_get_config_default(&acm_config);
acm_config.order = SLCD_AUTOMATED_CHAR_START_FROM_BOTTOM_RIGHT;
acm_config.fc = SLCD_FRAME_COUNTER_1;
acm_config.mode = SLCD_AUTOMATED_CHAR_SCROLL;
acm_config.seg_line_num = 3;
acm_config.start_seg_line = 4;
acm_config.row_digit_num = 5;
acm_config.digit_num = 5;
acm_config.scrolling_step = sizeof(display_array) - 5 + 1;
acm_config.com_line_num = 4;
acm_config.data_mask = 0xFF4002;
slcd_automated_char_set_config(&acm_config);

slcd_set_frame_counter(SLCD_FRAME_COUNTER_1,true,0x1E);
slcd_enable_frame_counter(SLCD_FRAME_COUNTER_1);

slcd_enable_automated_character();
slcd_enable();
delay_s(3);

/* 6. Automated Bit Mapping*/
slcd_disable();
slcd_disable_automated_character();
slcd_clear_display_memory();
slcd_disable_frame_counter(SLCD_FRAME_COUNTER_2);
slcd_set_automated_bit(ABM_MATRIX_SIZE,SLCD_FRAME_COUNTER_2);
slcd_set_frame_counter(SLCD_FRAME_COUNTER_2,true,0x1F);
slcd_enable_frame_counter(SLCD_FRAME_COUNTER_2);
slcd_enable_automated_bit();
slcd_enable();
```

9.1.2.2. Workflow

1. Display all case.

```
/* 1. Display all*/
slcd_set_display_memory();
delay_s(1);
slcd_clear_display_memory();
```

2. Display icon.

```
/* 2. Display icon*/
slcd_set_pixel(C42412A_ICON_USB);
slcd_set_pixel(C42412A_ICON_BAT);
slcd_set_pixel(C42412A_ICON_ATMEL);
delay_s(1);
```

3. Character map case.

```
/* 3. Character map*/
slcd_character_map_set(SLCD_AUTOMATED_CHAR_START_FROM_BOTTOM_RIGHT,3);
for(uint32_t i = 0 ; i < 5 ; i++) {
    slcd_character_write_data(0,4+i*4,character_map[10+i],0xFF4002);
}
delay_s(2);
```

4. SLCD blinking.

```
/* 4. Blinking*/
slcd_disable();
struct slcd_blink_config blink_config;
slcd_blink_get_config_defaults(&blink_config);
blink_config.blink_all_seg = false;
```

```
slcd_blink_set_config(&blink_config);

for(uint32_t i=0; i<4; i++){
    slcd_set_blink_pixel(i,0);
    slcd_set_blink_pixel(i,1);
}

slcd_set_frame_counter(SLCD_FRAME_COUNTER_0,true,0x18);
slcd_enable_frame_counter(SLCD_FRAME_COUNTER_0);

slcd_enable_blink();
slcd_enable();
delay_s(2);
```

5. Automated character mapping.

```
/* 5. Automated Character Mapping*/
slcd_disable();
struct slcd_automated_char_config acm_config;
slcd_automated_char_get_config_default(&acm_config);
acm_config.order = SLCD_AUTOMATED_CHAR_START_FROM_BOTTOM_RIGHT;
acm_config.fc = SLCD_FRAME_COUNTER_1;
acm_config.mode = SLCD_AUTOMATED_CHAR_SCROLL;
acm_config.seg_line_num = 3;
acm_config.start_seg_line = 4;
acm_config.row_digit_num = 5;
acm_config.digit_num = 5;
acm_config.scrolling_step = sizeof(display_array) - 5 + 1;
acm_config.com_line_num = 4;
acm_config.data_mask = 0xFF4002;
slcd_automated_char_set_config(&acm_config);

slcd_set_frame_counter(SLCD_FRAME_COUNTER_1,true,0x1E);
slcd_enable_frame_counter(SLCD_FRAME_COUNTER_1);

slcd_enable_automated_character();
slcd_enable();
delay_s(3);
```

6. Automated bit mapping.

```
/* 6. Automated Bit Mapping*/
slcd_disable();
slcd_disable_automated_character();
slcd_clear_display_memory();
slcd_disable_frame_counter(SLCD_FRAME_COUNTER_2);
slcd_set_automated_bit(ABM_MATRIX_SIZE,SLCD_FRAME_COUNTER_2);
slcd_set_frame_counter(SLCD_FRAME_COUNTER_2,true,0x1F);
slcd_enable_frame_counter(SLCD_FRAME_COUNTER_2);
slcd_enable_automated_bit();
slcd_enable();
```

10. Document Revision History

Doc. rev.	Date	Comments
42605A	12/2015	Initial release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**